Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Уфимский университет науки и технологий»

Кафедра Высокопроизводительных вычислительных технологий и систем

	1	2	3	4	5	6	7	8	9	10
100										
90										
80										
70										
60										
50										
40										
30										
20										
10										
0										

РАЗРАБОТКА ИЕРАРХИИ КЛАССОВ ДЛЯ ПОСТРОЕНИЯ ЛАБИРИНТОВ В C#

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

«Программирование»

3952.333206.000 ПЗ

Группа МКН-217	Фамилия И.О.	Подпись	Дата	Оценка
Студент	******* *. *.			
Консультант	******* *. *.			
Принял	******* *. *.			

Уфа 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
« Уфимский университет науки и технологий»

Кафедра Высокопроизводительных вычислительных технологий и систем

ЗАДАНИЕ

на курсовую работу по дисциплине

«Программирование»

Студент: ******* *. *. Группа: МКН-217

Консультант: ******* *. *.

1. Тема курсовой работы

Разработка иерархии классов для построения лабиринтов в C#

2. Основное содержание

2.1. Разработать иерархию классов для построения лабиринтов и выполнить ее программную реализацию в виде приложения на C# с графическим интерфейсом.
2.2. Оформить пояснительную записку к курсовой работе.

3. Требования к оформлению материалов работы

Требования к оформлению пояснительной записки

Пояснительная записка к курсовой работе должна быть оформлена в соответствии с требованиями ГОСТ и содержать

- титульный лист,
- задание на курсовую работу,
- содержание,
- введение,
- заключение,
- список литературы,
- приложение, содержащее листинг разработанной программы, если таковая имеется.

Дата выдачи задания Дата окончания работы

"__" _____ 202_ г. "__" _____ 202_ г.

Консультант _____ ******* *. *.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектно-ориентированное программирование (ООП) определяется как технология создания сложного программного обеспечения, которая основана на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого типа (класса), а классы образуют иерархию с наследованием свойств.

Основное достоинство ООП – сокращение количества межмодульных вызовов и уменьшение объёмов информации, передаваемой между модулями, по сравнению с модульным программированием. Это достигается за счёт более полной локализации данных и интегрирования их с подпрограммами обработки, что позволяет вести практически независимую разработку отдельных частей (объектов) программы.

Целью данной работы является изучение основных принципов объектно-ориентированного программирования на примере создания иерархии классов и разработки приложения на языке C# для построения лабиринтов различных видов.

1. ТЕОРЕТИЧЕСКИЕ СВДЕНИЯ

Лабиринт — структура в двухмерном или трехмерном пространстве, состоящая из запутанных путей к выходу

Имеется много способов построения лабиринтов, но для демонстрации работы ООП, достаточно будет разбираемых далее алгоритмов из теории графов и их модификаций, при помощи которых мы сможем получить образцовые лабиринты различных типов.

Лабиринты могут подразделятся на различные типы в зависимости от способа их генерации и прохождения, а также характеристик пространства, на котором они построены.

Мы будем разбирать двухмерные лабиринты в евклидовом пространстве которые реализованы в стандартной прямоугольной сетке, в которой ячейки имеют проходы, пересекающиеся под прямыми углами.

Разбираемые и реализуемые тут типы: Идеальный, плетеный, одномаршрутный.

1.1. Идеальный лабиринт

Идеальный лабиринт это такой, в котором нет петель или замкнутых цепей и не имеется недостижимых областей. Также он называется лабиринтом с одиночным соединением (simply-connected Maze). Из каждой точки существует ровно один путь к любой другой точке. Лабиринт имеет только одно решение. С точки зрения программирования такой лабиринт можно описать как дерево, связующее множество ячеек или вершин.

Часто именно этот тип имеется в виду, когда говорят про лабиринты в повседневности. Как базу для последующих алгоритмов мы будем использовать поле в виде “клетки”, для построения лабиринтов из которой мы будем вырезать проходы.

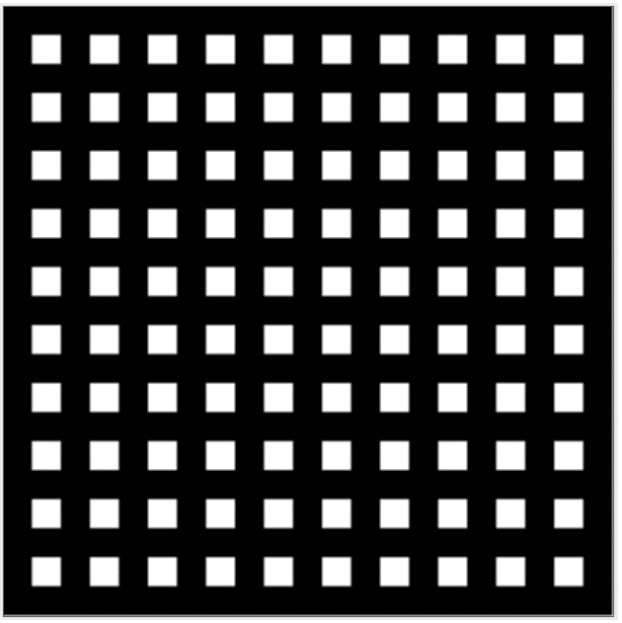

Рисунок 1 – База для будущего лабиринта

Для построения такого лабиринта мы будем использовать алгоритм Recursive backtracker: он требует стека, объём которого может доходить вплоть до размеров лабиринта. При вырезании путей он всегда вырезает проход в не пройдённой алгоритмом части, если она существует рядом с текущей ячейкой. Каждый раз, когда мы перемещаемся к новой ячейке, записываем предыдущую ячейку в стек. Если рядом с текущей позицией нет несозданных ячеек, то извлекаем из стека предыдущую позицию. Лабиринт завершён, когда в стеке больше ничего не остаётся. Это приводит к созданию лабиринтов, где тупиков меньше, но они длиннее, а решение обычно оказывается очень долгим и извилистым. При правильной реализации он выполняется быстро, и быстрее работают только очень специализированные алгоритмы.

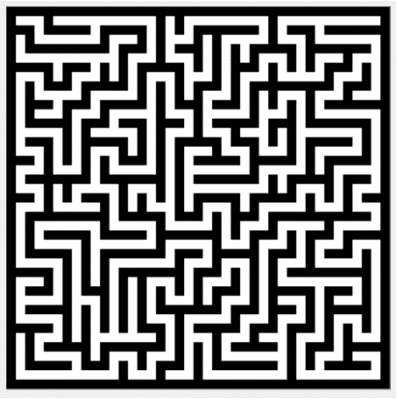

Рисунок 2 – Пример идеального лабиринта

1.2. Плетенный лабиринт

В плетенном лабиринте нет тупиков. Также его называют лабиринтом с многократными соединениями (purely multiply connected Maze). В таком лабиринте используются проходы, замыкающиеся и возвращающиеся друг к другу (отсюда название «плетёный»), они заставляют тратить больше времени на ходьбу кругами вместо попадания в тупики. Качественный плетёный лабиринт может быть гораздо сложнее идеального лабиринта того же размера.

Для генерации такого лабиринта для начала мы генерируем идеальный лабиринт, после чего ищем “тупики”, места в которых у клетки 3 стены. Их можно обрабатывать по разному, но мы используем самый простой способ для достижения необходимого лабиринта, удаляем случайно одну из стен у каждого из тупиков по очереди, в случае если после прошлых изменений один из тупиков перестал им быть мы удаляем его из списка тупиков. Тем самым мы избавляемся от тупиков достигая необходимого нам результата.

Благодаря возможности по-разному обрабатывать тупики мы можем сделать частично плетеный лабиринт (в нем есть и петли, и тупики) отбросив часть тупиков при обработке, или сделать петли длиннее выбирая стену не случайно.

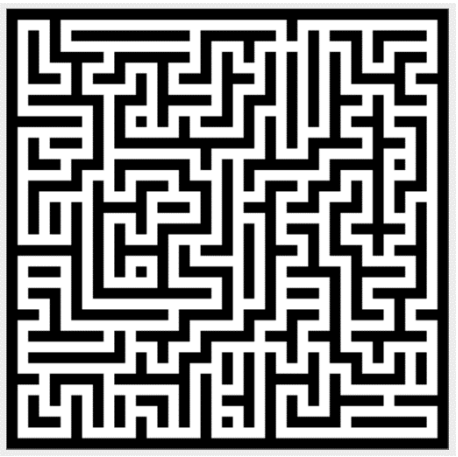

Рисунок 3 – Пример плетенного лабиринта

1.3. Одномаршрутный лабиринт

Под одномаршрутным подразумевается лабиринт без развилок. Одномаршрутный лабиринт содержит один длинный извивающийся проход, который меняет направление на всём протяжении лабиринта. Он

не очень сложен для прохождения, только если вы случайно не повернёте назад на полпути и не вернётесь в начало.

При его генерации мы модифицируем базовую клетку, увеличивая размер пустой клетки в 3 раза по сравнению со стенами, потом по модифицированной клетке строим идеальный лабиринт, и по середине дороги строим новую стену разрушая ее лишь в начале или конце пути. Тем самым мы формируем лабиринт, в котором есть единственная дорога.

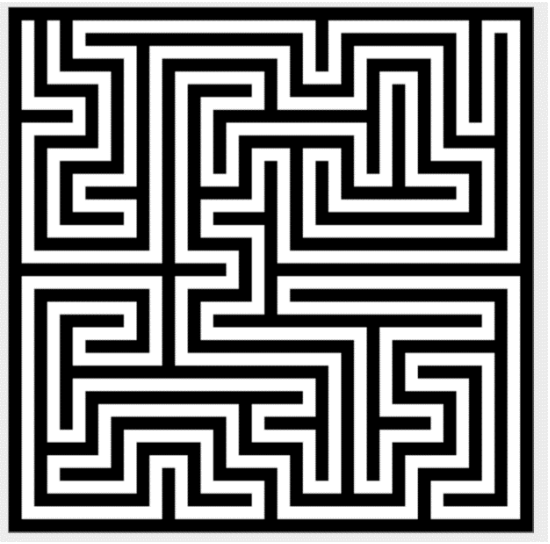

Рисунок 4 – Пример одномаршрутного лабиринта

1.4. Рандомный лабиринт и поиск пути

Также у нас присутствует 4 тип лабиринта – рандомный, где стены выбираются случайно, он создан с целью демонстрации.

Той же цели служит и алгоритм поиска пути реализованный через поиск в ширину.

2. РАЗРАБОТКА ИЕРАРХИИ КЛАССОВ

Перед тем как приступить к программной реализации приложения, необходимо разработать иерархию классов. Диаграмма классов приведена на рис. 1.

Базовым классом является класс CellMapPicture. От класса CellMapPicture наследуются класс LabirintBase. Класс Light является классом-родителем для классов SingleRouteLabirin, PerfectLabirint, RandomLabirint и WickerLabirint. Также для управления и визуализации есть класс Form1, а также класс, запускающий программу Program.

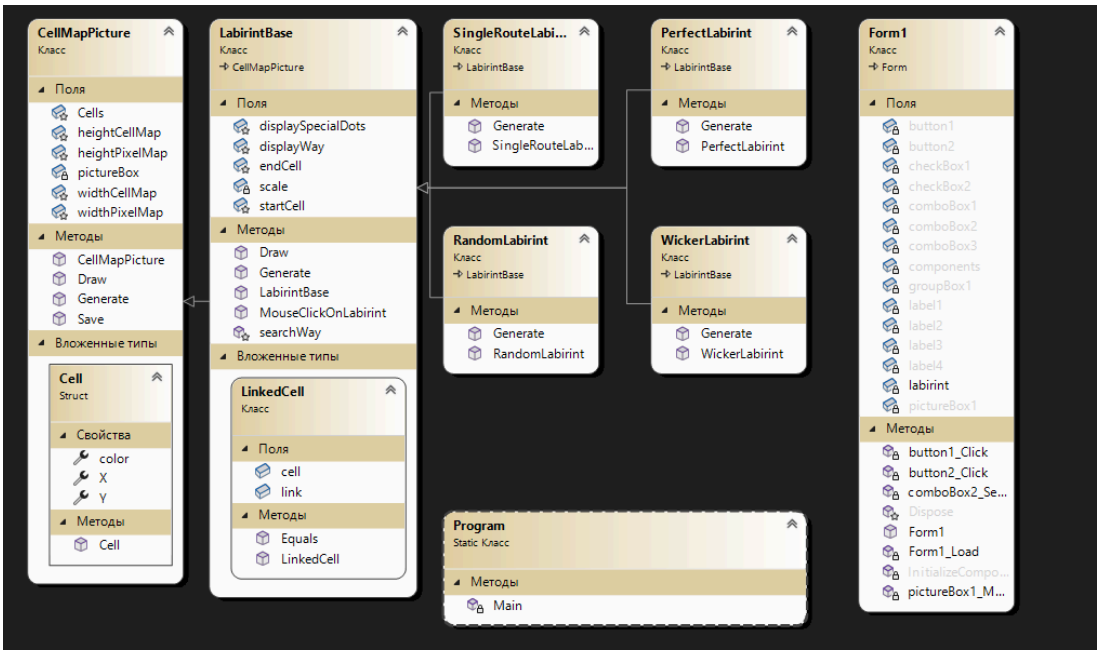

Рисунок 5 –Диаграмма классов

2.1.Класс CellMapPicture

Класс CellMapPicture содержит в себе поля и методы предназначенные для работы с базовой структурой панели из ячеек, а также вложенную структуру с описанием свойств ячейки и ее конструктором.

Поля класса:

- Cells – Контейнер для ячеек
- heightCellMap, widthCellMap – размер поля в ячейках
- heightPixelMap, widthPixelMap – размер поля в пикселях
- pictureBox – хранит ссылку на окно в котором все отображается

Методы класса:

- CellMapPicture – Конструктор
- Draw– метод отрисовывающий наши ячейки
- Generate – Метод генерации нового заполнения ячейками.
- Save – сохраняет полученную картинку

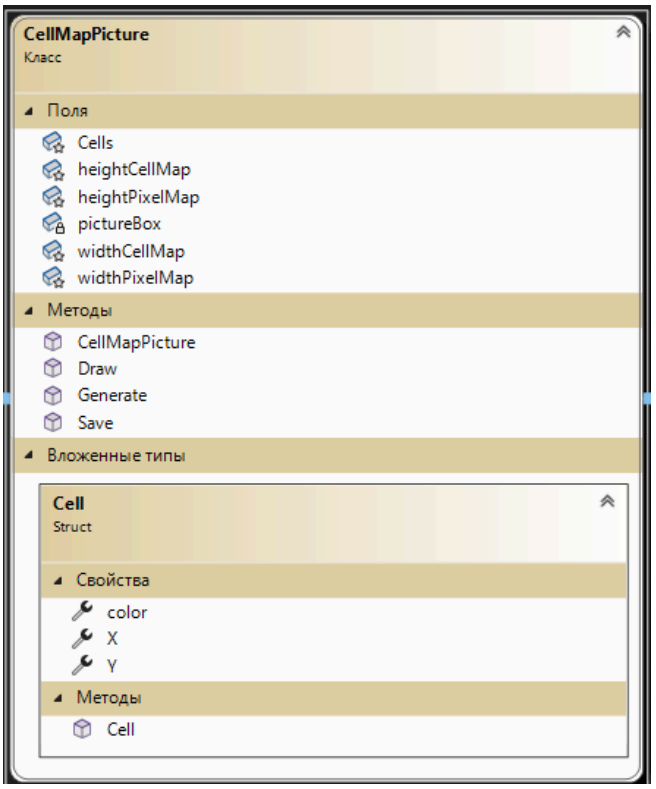

Рисунок 6 – класс CellMapPicture

Поля и метод вложенной структуры:

- Color – поле содержащие информацию о цвете ячейки
- X – поле с координатой по X среди клеток
- Y – поле с координатой по Y среди клеток

2.2. Класс LabirintBase

Данный класс является ключевым, так как реализует логику, стоящую за каждым лабиринтом, в частности базовую клетку, поиск пути, обработку нажатий и параметры отрисовки. Содержит вложенный класс, предназначенный для корректной работы поиска пути.

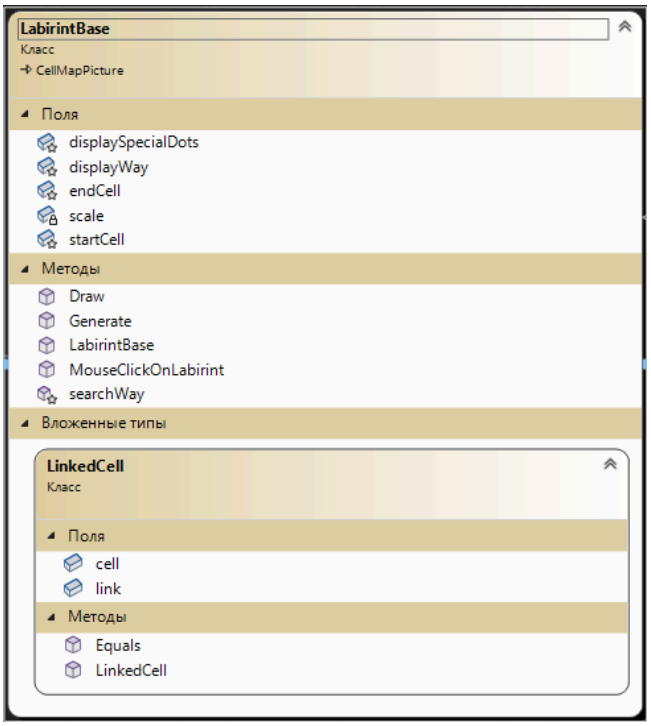

Рисунок 7 – Класс LabirintBase

Поля класса:

- DisplaySpecialDots –отображать ли начало и конец пути?
- DisplayWay – отображать ли Путь?
- endCell, startCell – Начальная и конечная ячейки пути.

Методы класса:

- Draw – Переопределенный метод отрисовывающий наши ячейки.
- Generate – Переопределенный метод генерации нового заполнения ячейками.
- LabirintBase – Конструктор.
- MouseClickOnLabirint – обработчик нажатий мышки.
- SearchWay – метод поиска пути.

Поля и методы вложенной структуры:

- cell – поле содержащее ячейку.
- link – поле содержащее ссылку на связанный LinkedCell.
- Equals – переопределенный метод сравнения.
- LinckedCell – конструктор.

2.3. Классы SingleRouteLabirin, PerfectLabirint, RandomLabirint и WickerLabirint

Все эти классы имеют схожий функционал, но реализуют разные виды лабиринтов.

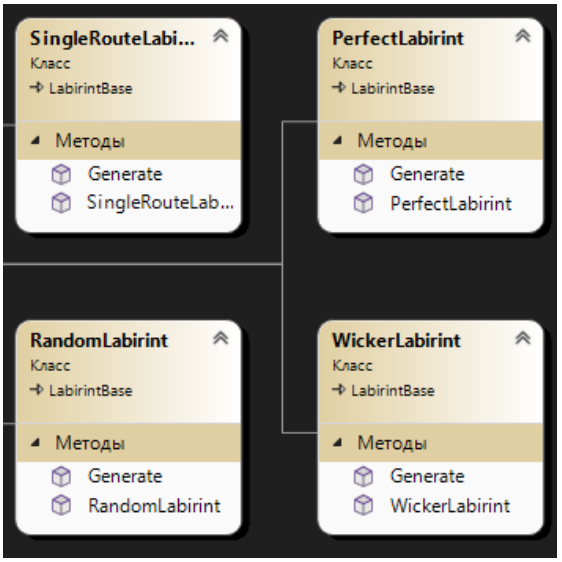

Рисунок 8 – Классы лабиринтов

Методы классов:

- Generate – Переопределение генерации, у каждого класса свои особенности генерации.
- SingleRouteLabirin, PerfectLabirint, RandomLabirint и WickerLabirint – Конструкторы классов.

2.4. Класс Program

Данный класс отвечает за запуск нашего решения (в частности формы), так как содержит метод main

Рисунок 9 – Класс Program

Методы класса:

- Main – Начальная точка программы

2.5.Класс Form1

Данный класс отвечает за визуальное взаимодействие с программой посредством пользовательского интерфейса.

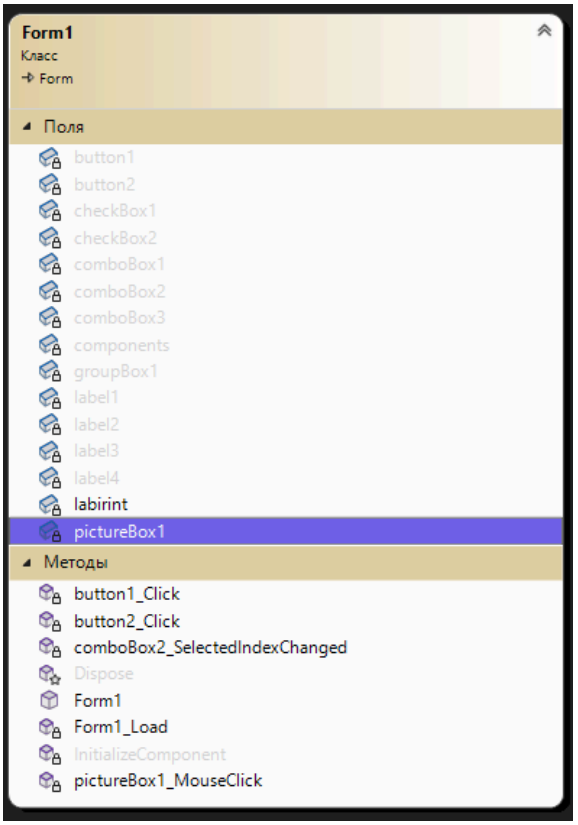

Рисунок 10 – Класс Form1

Поля класса:

- labirint – поле типа LabirintBase, в котором упаковывается созданный лабиринт.
- Все остальные – компоненты формы для взаимодействия с пользователем.

Методы класса:

- Все методы – Стандартные методы формы и методы для считывания событий, вызванных компонентами формы

3. ОПИСАНИЕ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ИЕРАРХИИ КЛАССОВ

Опишем работу разработанной программы, для реализации иерархии классов. Интерфейс программы состоит из кнопок «Сгенерировать», «Сохранить», которые генерируют и сохраняют выбранный лабиринт, комбо-боксов «X», «Y», регулирующие размер лабиринта. Также в интерфейсе программы есть чек-боксы «Отображение точек» и «Отображение пути», которые настраивают создание и отображение ключевых точек или пути, а также еще один комбо-бокс для выбора типа лабиринта. с

На рис. 11 показано начальное состояние приложения.

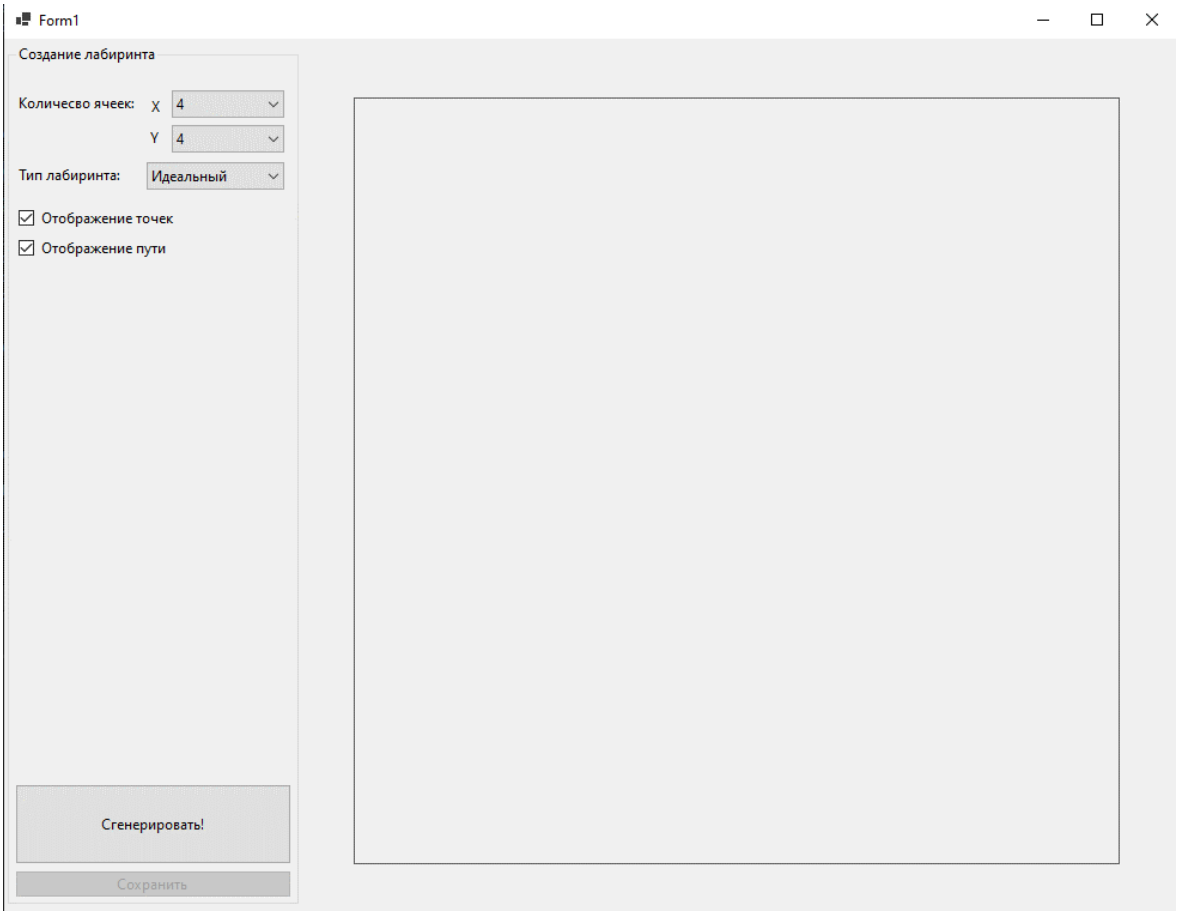

Рисунок 11 – Вид приложения при запуске

Пример 1. Построение идеального лабиринта не квадратной формы показано на рис. 12.

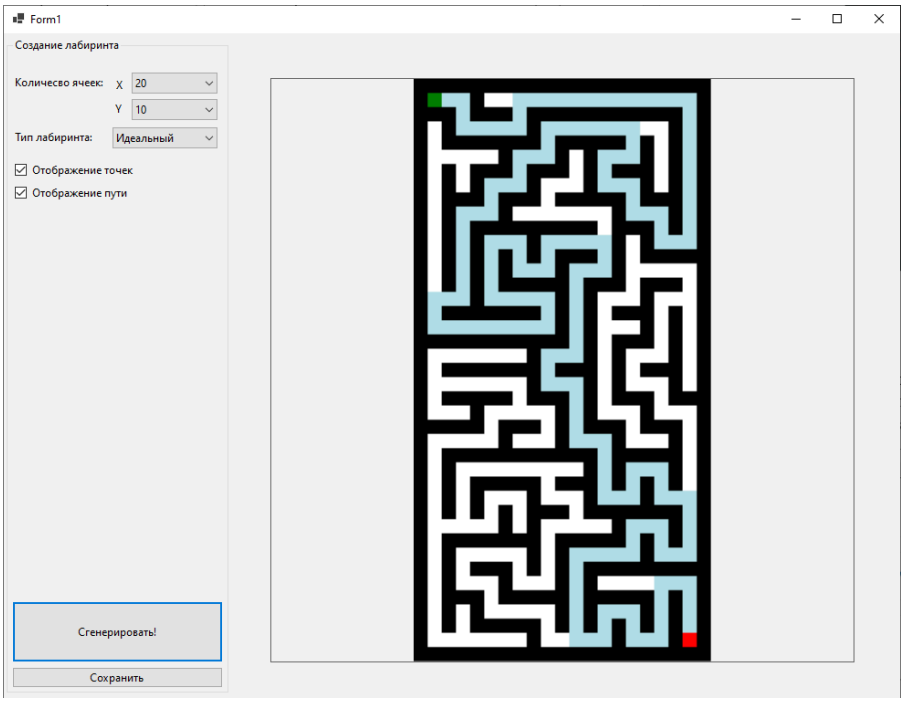

Рисунок 12 – Пример построения лабиринта по заданным параметрам

Пример 2. Построение плетенного лабиринта не квадратной формы показано на рис. 13.

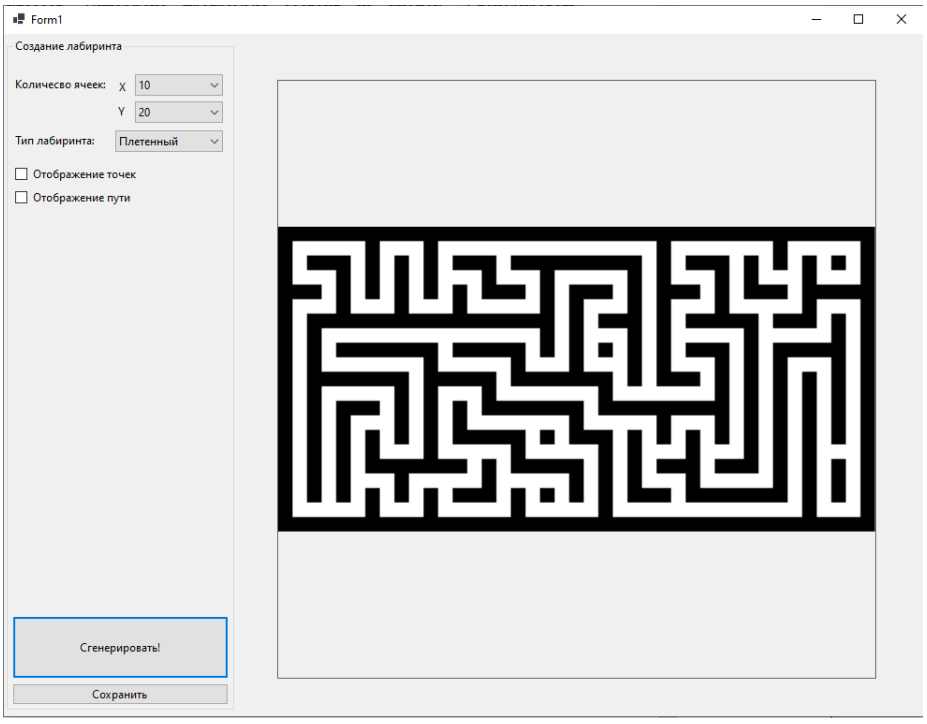

Рисунок 13 – Пример построения лабиринта по заданным параметрам

Пример 3. Построение одномаршрутного лабиринта квадратной формы показано на рис. 14.

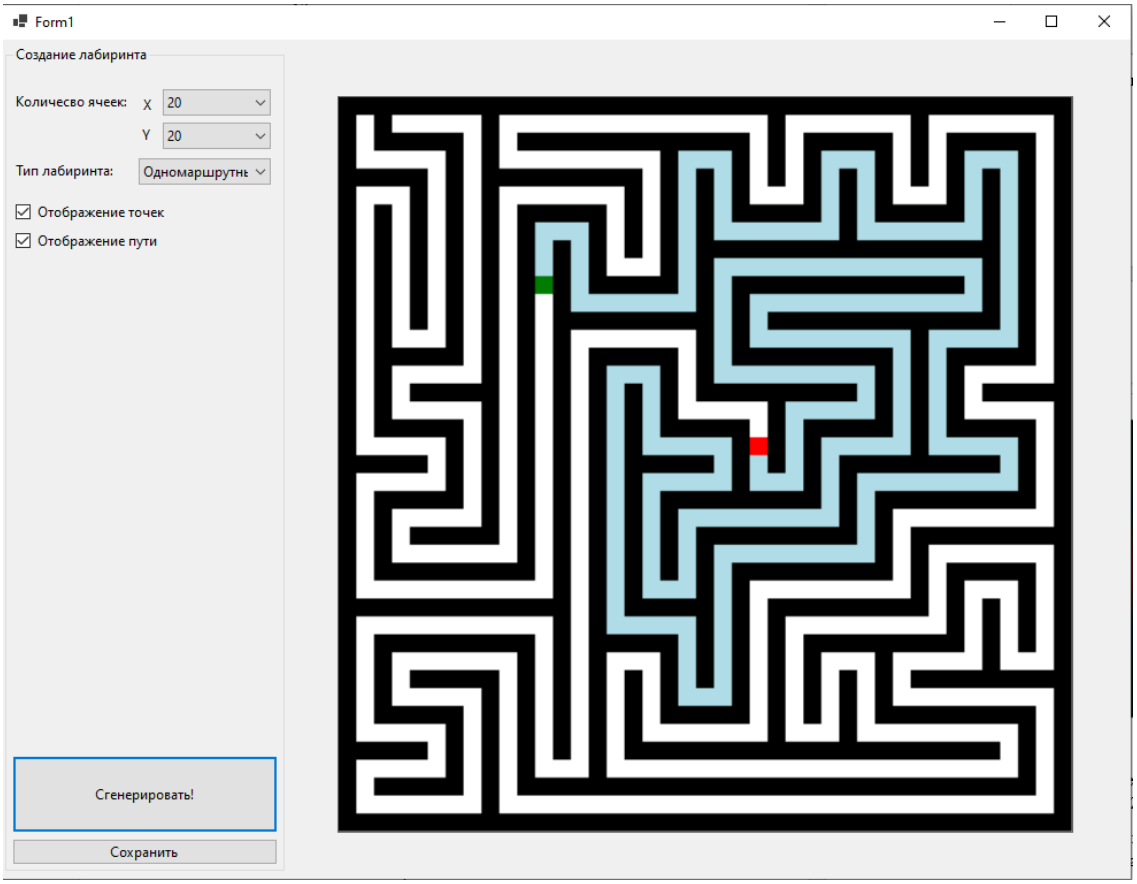

Рисунок 14 – Одномаршрутный лабиринт с нестандартными точками, выбранными мышью.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были изучены способы генерации различных лабиринтов и нахождения пути их прохождения.

Реализованы различные алгоритмы построения лабиринтов: идеальный, плетеный, одномаршрутный.

Для этой задачи была разработана иерархия классов и выполнена ее программная реализация с графическим интерфейсом на языке C#.

СПИСОК ЛИТЕРАТУРЫ

1. Buck, J., Mazes for Programmers: Code Your Own Twisty Little Passages. – “Pragmatic Bookshelf” 2015. – 288 с.
2. Иванова, Г.С., Ничушкина, Т.Н., Пугачев, Е.К. Объектно-ориентированное программирование: Учеб. для вузов / Под ред. Г.С. Ивановой. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001. – 320 с.
3. Троелс, Э. Джепикс, Ф. Язык программирования C# 7 и платформы .NET и .NET Core. / Э. Троелс, Ф. Джепикс. – СПб.: ООО “Диалектика”, 2018. – 1328 с.
4. Лабиринты: классификация, генерирование, поиск решений: коллективный блог. – 2019. – URL: https://habr.com/ru/post/445378/ (дата обращения: 15.11.22).

ПРИЛОЖЕНИЕ А
(обязательное)
ЛИСТИНГ ПРОГРАММЫ

Program.cs

```
namespace Labirint
{
    internal static class Program
    {
        [STAThread]
        static void Main()
        {
            ApplicationConfiguration.Initialize();
            Application.Run(new Form1());
        }
    }
}
```

Form.cs

```
using Labirint;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Text;
using System.Runtime.CompilerServices;
using System.Security;
using System.Windows.Forms;
using Labirint.LabirintsClasses;
using static System.Net.Mime.MediaTypeNames;

namespace Labirint
{
    public partial class Form1 : Form
    {
        LabirintBase labirint;
        public Form1()
        {
            InitializeComponent();
            comboBox1.SelectedIndex = 0;
            comboBox2.SelectedIndex = 0;
            comboBox3.SelectedIndex = 0;
        }
        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (comboBox2.SelectedItem.ToString() == "Идеальный")
            {
                labirint = new PerfectLabirint(Convert.ToInt32(comboBox1.SelectedItem),
Convert.ToInt32(comboBox3.SelectedItem), 10, pictureBox1, checkBox1.Checked, checkBox2.Checked);
            }
            else if (comboBox2.SelectedItem.ToString() == "Плетенный")
            {
                labirint = new WickerLabirint(Convert.ToInt32(comboBox1.SelectedItem),
Convert.ToInt32(comboBox3.SelectedItem), 10, pictureBox1, checkBox1.Checked, checkBox2.Checked);
            }
            else if (comboBox2.SelectedItem.ToString() == "Одномаршрутный"){
                labirint = new SingleRouteLabirint(Convert.ToInt32(comboBox1.SelectedItem),
Convert.ToInt32(comboBox3.SelectedItem), 10, pictureBox1, checkBox1.Checked, checkBox2.Checked);
```

```
            }
            else if (comboBox2.SelectedItem.ToString() == "Рандомные проходы")
            {
                labirint = new RandomLabirint(Convert.ToInt32(comboBox1.SelectedItem),
Convert.ToInt32(comboBox3.SelectedItem), 10, pictureBox1, checkBox1.Checked,checkBox2.Checked);
            }
            labirint.Generate();
            labirint.Draw();
            button2.Enabled = true;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            labirint.Save();
        }

        private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (comboBox2.SelectedItem.ToString() == "Идеальный")
            {
                checkBox1.Checked = true;
                checkBox1.Enabled = true;
                checkBox2.Checked = true;
                checkBox2.Enabled = true;
            }
            else if (comboBox2.SelectedItem.ToString() == "Плетенный")
            {
                checkBox1.Checked = true;
                checkBox1.Enabled = true;
                checkBox2.Checked = true;
                checkBox2.Enabled = true;
            }
            else if (comboBox2.SelectedItem.ToString() == "Одномаршрутный")
            {
                checkBox1.Checked = true;
                checkBox1.Enabled = true;
                checkBox2.Checked = true;
                checkBox2.Enabled = true;
            }
            else if (comboBox2.SelectedItem.ToString() == "Рандомные проходы")
            {
                checkBox1.Checked = true;
                checkBox1.Enabled = true;
                checkBox2.Checked = false;
                checkBox2.Enabled = false;
            }
        }

        private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
        {
            if(labirint != null)
            {
                labirint.MouseClickOnLabirint(pictureBox1, e);
            }
        }
    }
}
```

CellMapPicture.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace Labirint.LabirintsClasses
{
    internal class CellMapPicture
    {
        protected struct Cell
        {
            public int X { get; }
            public int Y { get; }
            public Color color { get; set; }
            public Cell(int LocalX, int LocalY, Color color)
            {
                this.X = LocalX;
                this.Y = LocalY;
                this.color = color;
            }
        }
        PictureBox pictureBox;
        protected int heightCellMap;
        protected int widthCellMap;
        protected int heightPixelMap;
        protected int widthPixelMap;
        protected Cell[,] Cells;
        public CellMapPicture(int heightCellMap, int widthCellMap, int scale, PictureBox pictureBox)
        {
            this.heightCellMap = heightCellMap;
            this.widthCellMap = widthCellMap;
            this.Cells = new Cell[this.widthCellMap, this.heightCellMap];
            this.pictureBox = pictureBox;
            this.heightPixelMap = this.heightCellMap * scale;
            this.widthPixelMap = this.widthCellMap * scale;
        }

        public virtual void Generate()
        {
            for (int i = 0; i < widthCellMap; i++)
            {
                for (int j = 0; j < heightCellMap; j++)
                {
                    Cells[i, j] = new Cell(i, j, i % 2 + j % 2 == 1 ? Color.Black : Color.White);
                }
            }
        }
        public virtual void Draw()
        {
            pictureBox.Image = new Bitmap(widthPixelMap, heightPixelMap);
            Graphics g = Graphics.FromImage(pictureBox.Image);
            Pen pen = new Pen(Color.Aqua, 5f);
            g.FillRectangle(new SolidBrush(Color.Black), 0, 0, widthPixelMap, heightPixelMap);
            for (int i = 0; i < widthCellMap; i++)
            {
                for (int j = 0; j < heightCellMap; j++)
                {
                    g.FillRectangle(new SolidBrush(Cells[i, j].color),
                        (int)((widthPixelMap / (float)widthCellMap) * Cells[i, j].X),
                        (int)((heightPixelMap / (float)heightCellMap) * Cells[i, j].Y),
                        (int)(widthPixelMap / (float)widthCellMap),
                        (int)(heightPixelMap / (float)heightCellMap));
                }
            }
            pictureBox.Update();
```

```
        }
        public void Save()
        {
            SaveFileDialog savedialog = new SaveFileDialog();
            savedialog.Title = "Сохранить картинку как...";
            savedialog.OverwritePrompt = true;
            savedialog.CheckPathExists = true;
            savedialog.Filter = "Image Files(*.BMP)|*.BMP|Image Files(*.JPG)|*.JPG|Image
Files(*.GIF)|*.GIF|Image Files(*.PNG)|*.PNG|All files (*.*)|*.*";
            if (savedialog.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    pictureBox.Image.Save(savedialog.FileName, System.Drawing.Imaging.ImageFormat.Png);
                }
                catch
                {
                    MessageBox.Show("Невозможно сохранить изображение", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }


    }
}
```

LabirintBase.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.NetworkInformation;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Labirint.LabirintsClasses
{
    internal class LabirintBase : CellMapPicture
    {
        protected Cell startCell;
        protected Cell endCell;
        protected bool displaySpecialDots;
        protected bool displayWay;
        int scale;
        public LabirintBase(int heightCellMap, int widthCellMap, int scale, PictureBox pictureBox, bool
displaySpecialDots, bool displayWay)
        : base((heightCellMap * 2 + 1), (widthCellMap * 2 + 1), scale, pictureBox)
        {
            this.displaySpecialDots = displaySpecialDots;
            this.displayWay = displayWay;
            this.scale = scale;
        }
        public override void Generate()
        {

            for (int i = 0; i < widthCellMap; i++)
            {
                for (int j = 0; j < heightCellMap; j++)
                {
                    Cells[i, j] = new Cell(i, j, i % 2 + j % 2 == 2 ? Color.White : Color.Black);
                }
            }
```

```
    startCell = Cells[1, 1];
    endCell = Cells[widthCellMap - 2, heightCellMap - 2];
}
public void MouseClickOnLabirint(PictureBox pb, MouseEventArgs e)
{
    double k = Math.Min(1.0 * pb.Width / widthPixelMap, 1.0 * pb.Height / heightPixelMap);
    Cell bufCell = Cells[(int)((e.Location.X - (pb.Width / 2 - widthPixelMap / 2 * k)) / k) / scale,
(int)((e.Location.Y - (pb.Height / 2 - heightPixelMap / 2 * k)) / k) / scale];
    if (bufCell.color != Color.Black)
    {
        foreach (Cell cl in Cells)
        {
            if (cl.color != Color.Black)
            {
                Cells[cl.X, cl.Y].color = Color.White;
            }
        }
        if (e.Button == MouseButtons.Left)
        {
            startCell = bufCell;
        }
        else if (e.Button == MouseButtons.Right)
        {
            endCell = bufCell;
        }
        Draw();
    }

}
public override void Draw()
{
    if (displayWay)
    {
        searchWay();
    }
    if (displaySpecialDots)
    {
        Cells[startCell.X, startCell.Y].color = Color.Green;
        Cells[endCell.X, endCell.Y].color = Color.Red;
    }
    base.Draw();
}
class LinkedCell
{
    public LinkedCell(Cell cell, LinkedCell link)
    {
        this.cell = cell;
        this.link = link;
    }
    public Cell cell;
    public LinkedCell link;
    public override bool Equals(object? obj)
    {
        if(cell.X == ((obj as LinkedCell).cell.X) && cell.Y == ((obj as LinkedCell).cell.Y))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

```
        protected void searchWay()
        {
            Stack<LinkedCell> CellVisited = new Stack<LinkedCell>();
            Queue<LinkedCell> waySearch = new Queue<LinkedCell>();
            waySearch.Enqueue(new LinkedCell(startCell, null));
            while (waySearch.Count != 0)
            {
                LinkedCell cell = waySearch.Dequeue();
                if(!CellVisited.Contains(cell))
                {
                    if (cell.cell.X < widthCellMap && Cells[cell.cell.X + 1, cell.cell.Y].color != Color.Black)
                    {
                        waySearch.Enqueue(new LinkedCell(Cells[cell.cell.X + 1, cell.cell.Y], cell));
                    }
                    if (cell.cell.Y < heightCellMap && Cells[cell.cell.X, cell.cell.Y + 1].color != Color.Black)
                    {
                        waySearch.Enqueue(new LinkedCell(Cells[cell.cell.X, cell.cell.Y + 1], cell));
                    }
                    if (cell.cell.X > 0 && Cells[cell.cell.X - 1, cell.cell.Y].color != Color.Black)
                    {
                        waySearch.Enqueue(new LinkedCell(Cells[cell.cell.X - 1, cell.cell.Y], cell));
                    }
                    if (cell.cell.Y > 0 && Cells[cell.cell.X, cell.cell.Y - 1].color != Color.Black)
                    {
                        waySearch.Enqueue(new LinkedCell(Cells[cell.cell.X, cell.cell.Y - 1], cell));
                    }
                    CellVisited.Push(cell);
                }
                if (cell.cell.X == endCell.X && cell.cell.Y == endCell.Y)
                {
                    break;
                }
            }
            Stack<Cell> way = new Stack<Cell>();
            LinkedCell LastCell = CellVisited.Pop();
            while (true)
            {
                LastCell.cell.color = Color.PowderBlue;
                way.Push(LastCell.cell);
                if (LastCell.cell.X == startCell.X && LastCell.cell.Y == startCell.Y)
                {
                    break;
                }
                LastCell = LastCell.link;
            }
            foreach (Cell cl in way)
            {
                Cells[cl.X, cl.Y] = cl;
            }
        }

    }
}
```

PerfectLabirint.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Labirint.LabirintsClasses
{
   internal class PerfectLabirint : LabirintBase
   {
      public PerfectLabirint(int heightCellMap, int widthCellMap, int scale, PictureBox pictureBox, bool
displaySpecialDots, bool displayWay)
   : base(heightCellMap, widthCellMap, scale, pictureBox,displaySpecialDots,displayWay)
      {
      }
      public override void Generate()
      {
         base.Generate();
         Random rnd = new Random();
         Stack<Cell> wayCells = new Stack<Cell>();
         Cell cl;
         bool[,] itsVisited = new bool[widthCellMap, heightCellMap];
         int numbVisited = 0;
         wayCells.Push(startCell);
         cl = wayCells.Peek();
         itsVisited[cl.X, cl.Y] = true;
         numbVisited++;
         while (((widthCellMap - 1) / 2) * ((heightCellMap - 1) / 2) > numbVisited)
         {
            cl = wayCells.Peek();
            List<Cell> cellForRandomPick = new List<Cell>();
            if (cl.X < widthCellMap - 2 && !itsVisited[cl.X + 2, cl.Y])
            {
               cellForRandomPick.Add(Cells[cl.X + 2, cl.Y]);
            }
            if (cl.Y < heightCellMap - 2 && !itsVisited[cl.X, cl.Y + 2])
            {
               cellForRandomPick.Add(Cells[cl.X, cl.Y + 2]);
            }
            if (cl.X > 1 && !itsVisited[cl.X - 2, cl.Y])
            {
               cellForRandomPick.Add(Cells[cl.X - 2, cl.Y]);
            }
            if (cl.Y > 1 && !itsVisited[cl.X, cl.Y - 2])
            {
               cellForRandomPick.Add(Cells[cl.X, cl.Y - 2]);
            }
            if (cellForRandomPick.Count != 0)
            {

               Cell pickedCell = cellForRandomPick[rnd.Next(cellForRandomPick.Count)];
               Cells[(cl.X + pickedCell.X) / 2, (cl.Y + pickedCell.Y) / 2].color = Color.White;
               wayCells.Push(pickedCell);
               numbVisited++;
               itsVisited[wayCells.Peek().X, wayCells.Peek().Y] = true;
            }
            else
            {
               wayCells.Pop();
            }
         }
      }
   }
}
```

RandomLabirint.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace Labirint.LabirintsClasses
{
    internal class RandomLabirint : LabirintBase
    {

        public RandomLabirint(int heightCellMap, int widthCellMap, int scale, PictureBox pictureBox, bool
displaySpecialDots, bool displayWay)
        : base(heightCellMap, widthCellMap, scale, pictureBox,displaySpecialDots, displayWay)
        {

        }
        public override void Generate()
        {
            base.Generate();
            Random rnd = new Random();
            for (int i = 1; i < widthCellMap - 1; i++)
            {
                for (int j = 1 + i % 2; j < heightCellMap - 1; j += 2)
                {
                    Cells[i, j] = new Cell(i, j, rnd.Next(2) == 1 ? Color.White : Color.Black);
                }
            }
        }
    }
}
```

SingleRoutLabirint.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Labirint.LabirintsClasses
{
    internal class SingleRouteLabirint : LabirintBase
    {
        public SingleRouteLabirint(int heightCellMap, int widthCellMap, int scale, PictureBox pictureBox, bool
displaySpecialDots, bool displayWay)
: base(heightCellMap, widthCellMap, scale, pictureBox,displaySpecialDots,displayWay)
        {
        }
        public override void Generate()
        {
            for (int i = 0; i < widthCellMap; i++)
            {
                for (int j = 0; j < heightCellMap; j++)
                {
                    Cells[i, j] = new Cell(i, j, i % 4>0 && j % 4>0 ? Color.White : Color.Black);
                }
            }
            Random rnd = new Random();
            Stack<Cell> wayCells = new Stack<Cell>();
            Cell cl;
            bool[,] itsVisited = new bool[widthCellMap, heightCellMap];
            int numbVisited = 0;
            startCell = Cells[2,2];
            wayCells.Push(startCell);
            cl = wayCells.Peek();
```

```
            itsVisited[cl.X, cl.Y] = true;
            numbVisited++;
            while ((((widthCellMap - 1) / 2) * ((heightCellMap - 1) / 2))/4 > numbVisited)
            {
                cl = wayCells.Peek();
                List<Cell> cellForRandomPick = new List<Cell>();
                if (cl.X < widthCellMap - 4 && !itsVisited[cl.X + 4, cl.Y])
                {
                    cellForRandomPick.Add(Cells[cl.X + 4, cl.Y]);
                }
                if (cl.Y < heightCellMap - 4 && !itsVisited[cl.X, cl.Y + 4])
                {
                    cellForRandomPick.Add(Cells[cl.X, cl.Y + 4]);
                }
                if (cl.X > 3 && !itsVisited[cl.X - 4, cl.Y])
                {
                    cellForRandomPick.Add(Cells[cl.X - 4, cl.Y]);
                }
                if (cl.Y > 3 && !itsVisited[cl.X, cl.Y - 4])
                {
                    cellForRandomPick.Add(Cells[cl.X, cl.Y - 4]);
                }
                if (cellForRandomPick.Count != 0)
                {

                    Cell pickedCell = cellForRandomPick[rnd.Next(cellForRandomPick.Count)];
                    Cells[(cl.X + pickedCell.X) / 2, (cl.Y + pickedCell.Y) / 2].color = Color.White;
                    Cells[(cl.X + pickedCell.X) / 2 + ((cl.Y - pickedCell.Y) != 0 ? 1 : 0), (cl.Y + pickedCell.Y) / 2 + ((cl.X -
pickedCell.X) != 0 ? 1 : 0)].color = Color.White;
                    Cells[(cl.X + pickedCell.X) / 2 - ((cl.Y - pickedCell.Y) != 0 ? 1 : 0), (cl.Y + pickedCell.Y) / 2 - ((cl.X -
pickedCell.X) != 0 ? 1 : 0)].color = Color.White;
                    wayCells.Push(pickedCell);
                    numbVisited++;
                    itsVisited[wayCells.Peek().X, wayCells.Peek().Y] = true;
                }
                else
                {
                    wayCells.Pop();
                }
            }
            for(int i = 2;i<heightCellMap-2;i++)
            {
                for(int j = 2;j<widthCellMap-2;j++)
                {
                    if ((i % 2 + j % 2) == 0) Cells[j, i].color = Color.Black;
                }
            }
            recursDel(2, 2);
            void recursDel (int x,int y)
            {

                if (x > 0 && x < widthCellMap && y < (heightCellMap - 3) && Cells[x, y + 2].color == Color.Black &&
Cells[x, y + 3].color == Color.White && Cells[x + 1, y + 2].color == Color.White &&
                    Cells[x - 1, y + 2].color == Color.White)
                {
                    Cells[x, y + 1].color = Color.Black;
                    recursDel(x, y + 2);
                }
                if (x > 0 && x < widthCellMap && y > 2 &&Cells[x, y - 2].color == Color.Black && Cells[x, y - 3].color
== Color.White && Cells[x + 1, y - 2].color == Color.White &&
                    Cells[x - 1, y - 2].color == Color.White)
                {
                    Cells[x, y - 1].color = Color.Black;
```

```
                recursDel(x, y - 2);
            }
            if (y > 0 && y < heightCellMap && x < (widthCellMap - 3) && Cells[x + 2, y].color == Color.Black &&
Cells[x + 3, y].color == Color.White && Cells[x + 2, y + 1].color == Color.White &&
                Cells[x + 2, y - 1].color == Color.White)
            {
                Cells[x + 1, y].color = Color.Black;
                recursDel(x + 2, y);
            }
            if (y > 0 && y < heightCellMap && x > 2 && Cells[x - 2, y].color == Color.Black && Cells[x - 3, y].color
== Color.White && Cells[x - 2, y + 1].color == Color.White &&
                Cells[x - 2, y - 1].color == Color.White)
            {
                Cells[x - 1, y].color = Color.Black;
                recursDel(x - 2, y);
            }

        }
        startCell = Cells[1, 1];
        endCell = Cells[3, 1];
        Cells[2, 1].color = Color.Black;

    }
  }
}
```

WickerLabirit.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Labirint.LabirintsClasses
{
   internal class WickerLabirint : LabirintBase
   {
      public WickerLabirint(int heightCellMap, int widthCellMap, int scale, PictureBox pictureBox, bool
displaySpecialDots, bool displayWay)
: base(heightCellMap, widthCellMap, scale, pictureBox, displaySpecialDots, displayWay)
      {
      }
      public override void Generate()
      {
         base.Generate();
         Random rnd = new Random();
         Stack<Cell> wayCells = new Stack<Cell>();
         Cell cl;
         bool[,] itsVisited = new bool[widthCellMap, heightCellMap];
         int numbVisited = 0;
         wayCells.Push(startCell);
         cl = wayCells.Peek();
         itsVisited[cl.X, cl.Y] = true;
         numbVisited++;
         List<Cell> deadEnds = new List<Cell>();
         while (((widthCellMap - 1) / 2) * ((heightCellMap - 1) / 2) > numbVisited)
         {
            cl = wayCells.Peek();
            List<Cell> cellForRandomPick = new List<Cell>();
            if (cl.X < widthCellMap - 2 && !itsVisited[cl.X + 2, cl.Y])
            {
               cellForRandomPick.Add(Cells[cl.X + 2, cl.Y]);
            }
            if (cl.Y < heightCellMap - 2 && !itsVisited[cl.X, cl.Y + 2])
            {
               cellForRandomPick.Add(Cells[cl.X, cl.Y + 2]);
            }
            if (cl.X > 1 && !itsVisited[cl.X - 2, cl.Y])
            {
               cellForRandomPick.Add(Cells[cl.X - 2, cl.Y]);
            }
            if (cl.Y > 1 && !itsVisited[cl.X, cl.Y - 2])
            {
               cellForRandomPick.Add(Cells[cl.X, cl.Y - 2]);
            }
            if (cellForRandomPick.Count != 0)
            {

               Cell pickedCell = cellForRandomPick[rnd.Next(cellForRandomPick.Count)];
               Cells[(cl.X + pickedCell.X) / 2, (cl.Y + pickedCell.Y) / 2].color = Color.White;
               wayCells.Push(pickedCell);
               numbVisited++;
               itsVisited[wayCells.Peek().X, wayCells.Peek().Y] = true;
            }
            else
            {
               wayCells.Pop();
```

```
    }
}
for (int i = 1; i <= widthCellMap-1; i+=2)
{
    for (int j = 1; j < heightCellMap-1; j+=2)
    {
        cl = Cells[i, j];
        int countWalls = 0;
        if (Cells[cl.X + 1, cl.Y].color == Color.Black)
        {
            countWalls++;
        }
        if (Cells[cl.X, cl.Y + 1].color == Color.Black)
        {
            countWalls++;
        }
        if (Cells[cl.X - 1, cl.Y].color == Color.Black)
        {
            countWalls++;
        }
        if (Cells[cl.X, cl.Y - 1].color == Color.Black)
        {
            countWalls++;
        }
        if(countWalls>2)
        {
            deadEnds.Add(Cells[cl.X, cl.Y]);
        }
    }
}
while(deadEnds.Count !=0)
{
    cl = deadEnds[0];
    List<Cell> cellForRandomPick = new List<Cell>();

    if (Cells[cl.X + 1, cl.Y].color == Color.Black && cl.X < widthCellMap - 2)
    {
        cellForRandomPick.Add(Cells[cl.X + 2, cl.Y]);
    }
    if (Cells[cl.X, cl.Y + 1].color == Color.Black && cl.Y < heightCellMap - 2)
    {
        cellForRandomPick.Add(Cells[cl.X, cl.Y + 2]);
    }
    if (Cells[cl.X - 1, cl.Y].color == Color.Black && cl.X > 1)
    {
        cellForRandomPick.Add(Cells[cl.X - 2, cl.Y]);
    }
    if (Cells[cl.X, cl.Y - 1].color == Color.Black && cl.Y > 1)
    {
        cellForRandomPick.Add(Cells[cl.X, cl.Y - 2]);
    }
    if(cellForRandomPick.Count == 0)
    {
        deadEnds.RemoveAt(0);
        continue;
    }
    else
    {
        Cell clq = new Cell();
        bool skipRnd = false;
        for(int i = 1; i < deadEnds.Count; i++)
        {
            for (int j = 0; j < cellForRandomPick.Count; j++)
```

```
                {
                    if (cellForRandomPick[j].X == deadEnds[i].X && cellForRandomPick[j].Y == deadEnds[i].Y)
                    {
                        clq = cellForRandomPick[j];
                        deadEnds.RemoveAt(i);
                        skipRnd = true;
                        break;
                    }
                }
            }
            if(!skipRnd)
            {
                clq = cellForRandomPick[rnd.Next(cellForRandomPick.Count)];
            }
            Cells[(cl.X + clq.X) / 2, (cl.Y + clq.Y) / 2].color = Color.White;
            deadEnds.RemoveAt(0);
        }
    }

}
}
}
```

ПЛАН-ГРАФИК

выполнения курсовой работы

обучающегося ******* *. *.

Наименование этапа работ	Трудоемкость выполнения, час.	Процент к общей трудоемкости выполнения	Срок предъявления консультанту
Получение и согласование задания	0,5	1,4	6 неделя
Знакомство с литературой по теме курсовой работы	8	22,2	7 неделя
Разработка иерархии классов	4	11,1	9 неделя
Реализация иерархии классов	16	44,4	12 неделя
Разработка и реализация интерфейса программы	2,5	6,9	13 неделя
Составление и оформление пояснительной записки и подготовка к защите	4,7	13,1	16 неделя
Защита	0,3	0,8	26 неделя
Итого	36	100	-